5장. 데이터 타입

5.1. 기초 데이터 타입

5.1.1. 숫자타입

- Int, Double 등은 클래스이며 이 클래스로 타입을 명시하여 선언한 변수는 그 자체로 객체

```
val intData : Int = 10
val result=intData.minus(5)
```

- Int, Double, Float, Long, Short, Byte, Char, Boolean, String, Any, Unit, Nothing 타입을 제공

Type	Bit width
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

5.1. 기초 데이터 타입

- 코틀린의 숫자 타입의 클래스들은 모두 Number 타입의 서브 클래스
- 문자(Characters)는 Number Type이 아니다.
- Number Type에 대한 자동 형 변형(implicit conversions for number)을 제공하지 않는다.

```
val a3: Byte=0b00001011
val a4: Int=123
val a5: Int=0x0F
val a6: Long = 10L
val a7: Double=10.0
val a8: Double=123.5e10
val a9: Float=10.0f
```

- 숫자 타입에 대입되는 데이터에 언더바(Underscore)를 추가 할수 있

```
val oneMillion: Int = 1_000_000
```

5.1. 기초 데이터 타입

5.1.2. 논리, 문자와 문자열 타입

```
val isTrue1: Boolean = true && false
val isTrue2: Boolean = true || false
val isTrue3: Boolean = !true
```

```
val charData='C'

fun check(c: Char) {
    if (c == 1) {//error

    }
}
```

```
var str: String = "Hello"
println("str[1] : ${str[1]}")
```

5.1. 기초 데이터 타입

문자열은 escaped string 과 raw string 으로 구분

```
val str2="Hello ₩n World"
val str3="""Hello
World"""
```

string template

```
println("result : $name .. ${sum(10)}")
```

5.1. 기초 데이터 타입

5.1.3. Any 타입

코틀린 클래스의 최상위 클래스가 Any

```
fun getLength(obj : Any) : Int {
    if(obj is String) {
        return obj.length
    }
    return 0
}

fun main(args: Array<String>) {
    println(getLength("Hello"))
    println(getLength(10))
}
```

```
fun cases(obj : Any): String{
    when(obj) {
        1 -> return "One"
        "Hello" -> return "Greeting"
        is Long -> return "Long"
        !is String -> return "Not a string"
        else -> return "unknown"
    }
}
```

5.1. 기초 데이터 타입

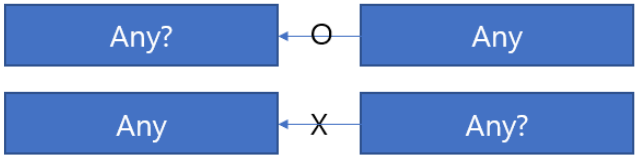
5.1.4. null 허용 타입

```
val a: Int =null//error
val b: Int? = null//ok~
```

```
fun parseInt(str: String): Int? {
    return str.toIntOrNull()
}
```

5.1. 기초 데이터 타입

5.1.5. Any, Any? 타입




```
val myVal1: Any = 10
val myVal2: Any?= myVal1

val myVal3: Any? = 10
val myVal4: Any = myVal3//error
val myVal5: Any = myVal3 as Any
```

```
val myInt1: Int = 10
val myInt2: Int? = myInt1

val myInt3: Int? = 10
val myInt4: Int = myInt3//error
val myInt5: Int = myInt3 as Int
```

5.1. 기초 데이터 타입

5.1.6. Unit 과 Nothing

```
fun myFun1(){ }
fun myFun2(): Unit { }
```

```
fun myFun(arg: Nothing?): Nothing {
    throw Exception()
}
```

5.1. 기초 데이터 타입

5.1.7. 타입확인과 캐스팅

- 타입 체크를 위해 is 연산자 이용
- is 연산자를 이용해 타입으로 체크되면 smart cast

```
fun getStringLength(obj: Any): Int? {
    val strData: String = obj//error
    if (obj is String) {
        return obj.length
    }
    return null
}
```

```
fun getStringLength2(obj: Any): Int? {
    if (obj !is String) return null
    return obj.length
}
```

5.1. 기초 데이터 타입

- 기초 데이터 타입에 대한 자동 형변형(implicit conversions for number)을 제공하지 않는다.
- 기초 타입의 캐스팅은 toXXX() 함수를 이용해 명시적으로 진행

```
var a1: Int =10
var a2 : Double = a1//error
```

```
var a1: Int =10
var a2 : Double = a1.toDouble()//ok~~
```

- toByte(): Byte
- toShort(): Short
- toInt(): Int
- toLong(): Long
- toFloat(): Float
- toDouble(): Double
- toChar(): Char

```
val l = 1L + 3 // Long + Int => Long
```

5.2. 컬렉션 타입

5.2.1. 배열

- Array로 표현되며 Array는 get, set. size등의 함수를 포함하고 있는 클래스

```
fun main(args: Array<String>) {
    //arrayOf 함수 이용
    var array = arrayOf(1, "kkang", true)
    array[0]=10
    array[2]="world"
    println("${array[0]} .. ${array[1]} .. ${array[2]}")
    println("size : ${array.size} .. ${array.get(0)} .. ${array.get(1)} .. ${array.get(2)}")

}
```

```
var arrayInt = arrayOf<Int>(10, 20, 30)
```

```
var arrayInt2= intArrayOf(10, 20, 30)
var arrayDouble= doubleArrayOf(10.0, 20.0, 30.0)
```

```
var array3=Array(3,  { i -> i*10})
```

```
var array4=Array<Int>(3, {i -> i * 10})
var array5=IntArray(3, { i -> i *10})
```

5.2. 컬렉션 타입

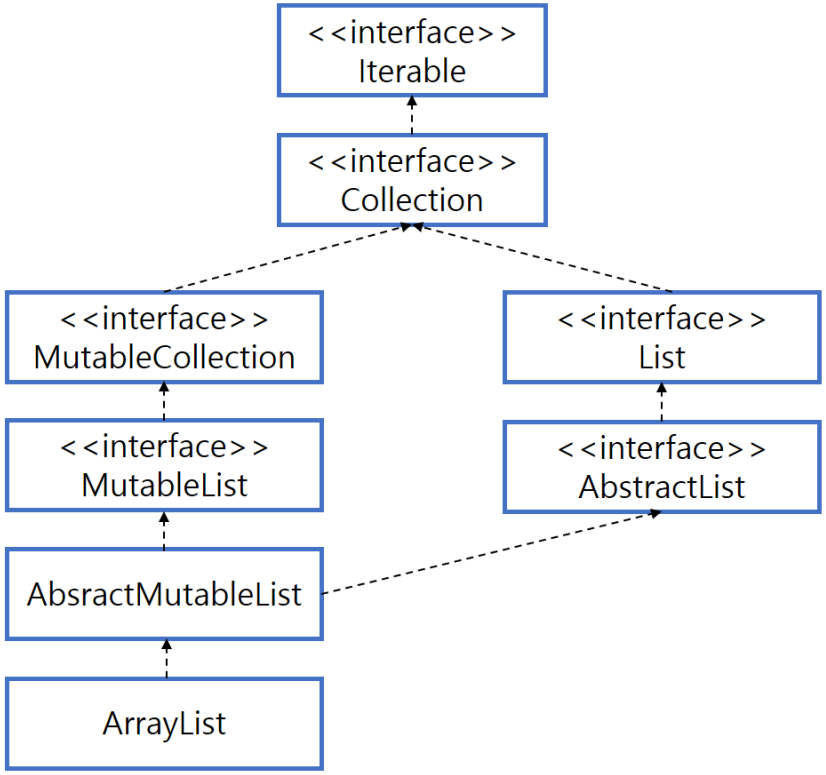
```
var array2= arrayOfNulls<Any>(3)
array2[0]=10
array2[1]="hello"
array2[2]=true
println("${array2[0]} .. ${array2[1]} .. ${array2[2]}")

var emptyArray=Array<String>(3,{""})
emptyArray[0]="hello"
emptyArray[1]="world"
emptyArray[2]="kkang"
println("${emptyArray[0]} .. ${emptyArray[1]} .. ${emptyArray[2]}")
```

5.2. 컬렉션 타입

5.2.2. List, Set, Map

- Collection 타입의 클래스들은 mutable 클래스와 immutable 클래스로 구분
- kotlin.collection.List 인터페이스로 표현되는 객체는 immutable 이며 size(), get() 함수만 제공
- kotlin.collection.MutableList 인터페이스로 표현되는 객체는 mutable 이며 size(), get() 함수 이외에 add(), set() 함수 제공



5.2. 컬렉션 타입

	타입	함수	특징
List	List	listOf()	Immutable
	MutableList	mutableListOf()	mutable
Map	Map	mapOf()	Immutable
	MutableMap	mutableMapOf()	mutable
Set	Set	setOf()	Immutable
	MutableSet	mutableSetOf()	mutable

```
fun main(args: Array<String>) {
    val immutableList: List<String> = listOf("hello", "world")
    println("${immutableList.get(0)} .. ${immutableList.get(1)}")
}
```

```
val mutableList: MutableList<String> = mutableListOf("hello", "world")
mutableList.add("kkang")
mutableList.set(1, "korea")
println("${mutableList.get(0)} .. ${mutableList.get(1)} .. ${mutableList.get(2)}")
```

```
val arrayList: ArrayList<String> = ArrayList()
arrayList.add("hello")
arrayList.add("kkang")
arrayList.set(1, "world")
println("${arrayList.get(0)} .. ${arrayList.get(1)}")
```

5.2. 컬렉션 타입

5.2.3. Iterfator

- hasNext() 함수와 next() 함수를 이용해 순차적 이용

```
val list1= listOf<String>("hello","list")
    val iterator1=list1.iterator()
    while (iterator1.hasNext()){
        println(iterator1.next())
    }
```